CENTRUL DE PREGĂTIRE PENTRU PERFORMANŢĂ HAI LA OLIMPIADĂ !

DISCIPLINA INFORMATICĂ

JUDEŢUL SUCEAVA

Titlul lectiei: Metoda backtracking

Data: 17.12.2016

Profesor: Ilincăi Florin

Grupa: clasa a X-a locul de desfasurare: CN PR Suceava

Metoda Backtracking

Această tehnică se aplică pentru problemele care îndeplinesc simultan următoarele condiţii:

- soluţia poate fi reprezentată sub forma unui vector $S=x_1,x_2,...,x_n$, cu $x_1 \in A_1$, $x_2 \in A_2$, ..., $x_n \in A_n$
- mulţimile A_1, A_2, ... ,A_n sunt mulţimi finite, iar elementele lor se consideră că se află într-o relație de ordine bine stabilită.

Observaţii:

1. nu pentru toate problemele n se cunoaşte de la început;
2. $x_1,x_2,...,x_n$, pot fi la rândul lor vectori;
3. în multe probleme A_1, A_2, ... ,A_n coincid.

Tehnica backtracking are ca rezultat obţinerea tuturor soluţiilor problemei. În cazul în care se dorește obţinerea unei singure soluţii se poate forţa oprirea atunci când aceasta a fost găsită.

Construirea soluţiei:

Tehnica foloseşte o rutină unică, aplicabilă fiecărei probleme, rutină construită cu ajutorul stivei. Rutina apelează*funcţii* care au acelaşi nume şi care, din punct de vedere al rutinei, realizează acelaşi lucru. Soluţia se generează sub forma de vector. Considerăm că generarea soţiilor se face într-o stivă. Astfel $x_1 \in A_1$, se va găsi pe primul nivel al stivei, $x_2 \in A_2$ se va găsi pe al doilea nivel aş stivei, ... $x_k \in A_k$ se găseşte pe nivelul k al stivei notată st.

Nivelul k+1 al stivei trebuie iniţializat (pentru a alege, în ordine, elementele mulţimii k+1).

Iniţializarea unui nivel al stivei se face cu o valoare aflată înaintea tuturor valorilor posibile din mulţime.

Procedura generală backtracking iterativă

```
void back()
{
int k=1;
st[k]=init(k);
```

```
while (k>0)
        {
        while (există o valoare netestată în Sk)
        {
        st[k]=urmator(k);
        if valid(st[1],st[2],...,st[k])
        if (k==n) solutie(st[1],st[2],...,st[n]);
                else {
                        k=k+1;
                        st[k]=init(k);
                        }
        }
        k=k-1;
}
```

1. **Generarea permutărilor**

Se citeşte un număr natural n. Se cere să se tipărească toate permutările mulţimii {1,2,…,n} în ordine lexicografică.

De exemplu, pentru n=3 avem 1 2 3, 1 3 2, 2 1 3, 2 3 1, 3 1 2, 3 2 1.

Definiţie: Permutările unei mulţimi A cu n elemente reprezintă toate modurile de aranjare a elementelor mulţimii A astfel încât fiecare element să fie scris o singură dată.

Reprezentarea soluţiei: - fiecare componentă are valori în mulţimea {1,2,….,n} -soluţia are n componente

Condiţia de validare: - fiecare componentă apare o singură dată

Numărul permutărilor: n! = 1*2*...*n

```
#include <iostream>
using namespace std;
int st[30];
int n,nr;
void tipar()
{
int i;
for(i=1;i<=n;i++)
   cout<<st[i]<<' ';
   cout<<'\n';
nr++;
}
int valid(int k)
{
int i;
for(i=1;i<k;i++)
   if(st[i]==st[k])return 0;
return 1;
}
void back()
{
int i,k=1;
st[1]=0;
while(k>0)
{
   while(st[k]<n)
      {
      st[k]=st[k]+1;
      if(valid(k))
         if(k==n) tipar();
      else
         {k=k+1;
         st[k]=0;
         }
}
k=k-1;
}}
int main()
{
cin>>n;
back();
cout<<nr;
return 0;
}
```

Varianta recursivă:

```
#include<iostream>
using namespace std;
int st[30];
int n,nr;

void tipar()
{
int i;
for(i=1;i<=n;i++)
   cout<<st[i]<<' ';
cout<<'\n';
nr++;
}
int valid(int k)
{
int i;
for(i=1;i<k;i++)
    if(st[i]==st[k])return 0;
return 1;

.
```

2. Problema colorării hărţii

Fie o hartă cu n ţări numerotate de la 1 la n. Se generează toate variantele de colorare a acestei hărţi având la dispoziţie 4 culori notate cu A, B, C, D, astfel încât oricare două ţări vecine să nu fie colorate la fel. Scrieţi un program care pentru o hartă dată să genereze toate variantele de colorare cu 4 culori a hărţii, astfel încât oricare două ţări vecine să nu fie colorate la fel.

```
#include<iostream>
#include<fstream>
using namespace std;
int A[21][21];
char st[21];
int n,nr;
ifstream fin("harta.in");
void citire()
{
int i,j;
fin>>n;
for(i=1;i<=n;i++)
   for(j=1;j<=n;j++)
      fin>>A[i][j];
}
void tipar()
{int i;
nr++;
for(i=1;i<=n;i++)
   cout<<st[i];
cout<<'\n';
}
int valid(int k)
{
int i;
for(i=1;i<k;i++)
   if((A[i][k]==1)&&(st[i]==st[k]))
      return 0;
return 1;
}
void back(int k)
{
char c;
if(k==n+1) tipar();
else
for(c='A';c<='D';c++)
{   st[k]=c;
if(valid(k)) back(k+1);
}}
int main()
```

```
{
citire();
back(1);
cout<<nr<<" solutii";
fin.close();
return 0;
}
```

.

3. Plata unei sume cu banknote de valori date

Se dau o suma şi n tipuri de bancnote având valori de a1,a2,...,an lei. Se cer toate modalităţile de plată a sumei utilizând aceste bancnote. Presupunem că dispunem de un număr suficient de bancnote de fiecare tip.

Exemplu: Pentru Suma=5 , n=3 (trei tipuri de bancnote) cu valorile 1,2,3, soluţiile sunt: Sol 1: 1 de 2, 1 de 3 Sol 2: 1 de 1, 2 de 2 Sol 3: 2 de 1, 1 de 3 Sol 4: 3 de 1, 1 de 2 Sol 5: 5 de 1

Reprezentarea soluţiei:

- soluţia are n componente

-fiecare componentă st[k] are valori în mulţimea {0,1,2,….,Suma/ak}

Condiţia de validare:

- suma parţială s=a[1]*st[1]+a[2]*st[2]+…+a[k]*st[k] < Suma

-suma parţială s se actualizează la adăugarea sau eliminarea unei valori st[k].

```
#include<iostream>
using namespace std;
int st[101],a[101];
int n,s,Suma,nr;
void tipar(int k)
{
int i;nr++;
cout<<"Solutia "<<nr<<":"<<endl;
for(i=1;i<=k;i++)
{
if(st[i]>0)
cout<<st[i]<<" de "<<a[i]<<endl;
}
cout<<endl;
}
void back(int k)
{
if(s==Suma)tipar(k-1);
else
{
st[k]=-1;
while(st[k]*a[k]+s<Suma && k<n+1)
{
st[k]=st[k]+1;
s=s+st[k]*a[k];
back(k+1);
s=s-st[k]*a[k];
}
}
}

int main()
{
int i;
cout<<"Suma="; cin>>Suma;
cout<<"n="; cin>>n;
for(i=1;i<=n;i++)
{
cout<<"a["<<i<<"]=";
cin>>a[i];
}
back(1);
return 0;
}
```

4. Programare sesiune

Un student are de dat n examene numerotate de la 1 la n intr-o sesiune formata din m zile (m este cel putin de 2 ori mai mare decat n). Afisati toate modurile in care isi poate programa studentul examenele astfel incat sa nu dea 2 examene in zile consecutive si sa dea examenele in ordine de la 1 la n.

Ex:

n=3
m=6
Solutii:
010203
100203
102003
102030
(0 codifica zilele libere)

```
#include<fstream>
using namespace std;
ifstream fin("date.in");
ofstream fout("date.out");

int p[100];

void tipar(int st[], int n, int m)
{
     for(int i=1;i<=m;i++)
                    fout<<st[i];
                    fout<<endl;
}
int valid(int st[], int k, int n, int m)
{
   if(k>1)
   {
      if(st[k-1]*st[k]!=0) return 0;//2
examene alaturate
      if(st[k]>1)//vreau sa pun examen
        {  //caut examen cu 1 mai mic
           for(int i=1;i<k;i++)
if(st[i]==st[k]-1) return 1;
          return 0;
        }
   }
   if(st[1]>1) return 0;//nu pot incepe cu
>1
   if(st[k]==0)//vreau sa pun zi libera
   {
      int s=0;
      for(int i=1;i<=k;i++) if(st[i]==0) s++;
      if(s>m-n) return 0;//nu pot fi mai
multi de 0 de m-n
   }
   return 1;
}

void back(int st[], int n, int m, int k)
{
   int i;
   for(i=0;i<=n;i++)
      if(!p[i]||i==0) //examenele nu se
repeta
      {
         st[k]=i;
         p[i]=1;
         if(valid(st,k,n,m))
                  if (k==m)
                           tipar(st,n,m);
                  else

back(st,n,m,k+1);
         p[i]=0;
      }

}

int main()
{
   int st[100],n,m;
   fin>>n>>m;
   back(st,n,m,1);
   fin.close();
   fout.close();
   return 0;
}
```

5. Combinații de n cifre

Un program citeşte o valoare naturală nenulă impară pentru n şi apoi generează şi afişează în ordine crescătoare lexicografic toate combinațiile formate din n cifre care îndeplinesc următoarele proprietăți:

- încep şi se termină cu 0;
- modulul diferenței între oricare două cifre alăturate dintr-o combinație este 1.

```
#include <iostream>
using namespace std;

int n,st[21];

int modul(int n)
{
  if(n<0) n=-n;
  return n;
}

void afisare()
{
  for(int i=1;i<=n;i++)
    cout<<st[i];
  cout<<'\n';
}

void back(int k)
{
  for(int i=0;i<=n/2;i++)
  {
    st[k]=i;
    if(st[1]==0 && (k==1 ||
modul(st[k]-st[k-1])==1))
      if(k==n)
      {
        if(st[n]==0) afisare();
      }
      else back(k+1);
  }
}

int main()
{
  cin>>n;
  back(1);
  return 0;
}
```

Bactracking în plan

În variantă clasică aplicam metoda *backtracking* pentru rezolvarea problemelor în care soluția era reprezentată ca vector. Putem generaliza ideea căutării cu revenire şi pentru probleme în care căutarea se face „în plan". Pentru noi planul va fi reprezentat ca un tablou bidimensional.

Pentru a intui modul de funcționare a metodei *backtracking* în plan să ne imaginăm explorarea unei peşteri. Speologul porneşte de la intrarea în peşteră şi trebuie să exploreze în mod sistematic toate culoarele peşterii. Ce înseamnă „în mod sistematic"? În primul rând îşi stabileşte o ordine pentru toate direcțiile posibile de mişcare (de exemplu, N, NE, E, SE, S, SV, V, NV) şi întotdeauna când se găseşte într-un punct din care are mai multe culoare de explorat, alege direcțiile de deplasare în ordinea prestabilită. În al doilea rând, speologul va plasa marcaje pe culoarele pe care le-a explorat, pentru ca nu cumva să se rătăcească şi să parcurgă de mai multe ori acelaşi culoar (ceea ce ar conduce la determinarea eronată a lungimii peşterii).

În ce constă explorarea? Speologul explorează un culoar până când întâlneşte o intersecție sau până când culoarul se înfundă. Dacă a ajuns la o intersecție, explorează succesiv toate culoarele care pornesc din intersecția respectivă, în ordinea prestabilită a direcțiilor. Când un culoar se înfundă, revine la intersecția precedentă şi alege un alt culoar, de pe următoarea direcție (dacă există; dacă nu există, revine la intersecția precedentă ş.a.m.d.).

Vom nota prin `NrDirectii` o constantă care reprezintă numărul de direcții de deplasare, iar `dx`, respectiv `dy` sunt doi vectori constanți care reprezintă deplasările relative pe direcția `Ox`, respectiv pe direcția `Oy`, urmând în ordine cele `NrDirectii` de deplasare.

```
void Bkt_Plan(int x, int y)
              //x, y reprezinta coordonatele pozitiei curente
{
Explorare(x,y);                  //exploram pozitia curenta
if (EFinal(x,y))          //pozitia x,y este un punct final
   Prelucrare_Solutie();
```

```
    else                                    //continuam cautarea
    for (i=0; i<NrDirectii; i++)
     //ma deplasez succesiv pe directiile posibile de miscare
     if (Nevizitat(x+dx[i], y+dy[i]))
                        //nu am mai trecut prin aceasta pozitie
                Bkt_Plan(x+dx[i], y+dy[i]);
}
```

Labirint

Într-un labirint, reprezentat ca o matrice `L`, cu `n` linii şi `m` coloane, cu componente `0` sau `1`, (`1` semnificând perete, `0` culoar) se găseşte o bucată de brânză pe poziția (`xb`, `yb`) şi un şoricel pe poziția (`xs`, `ys`). Afişați toate posibilitățile şoricelului de a ajunge la brânză, ştiind că el se poate deplasa numai pe culoar, iar direcțiile posibile de mişcare sunt N, NE, E, SE, S, SV, V, NV.

De exemplu, pentru un labirint cu `4` linii şi `4` coloane de forma următoare, în care şoricelul se găseşte pe poziția `1 1`, iar brânza pe poziția `4 4`

```
0 1 1 1
0 1 1 1
0 1 0 0
1 0 1 0
```

programul va afişa:

```
Solutia nr. 1
*111
*111
*1**
1*1*
Solutia nr. 2
*111
*111
*1*0
1*1*
```

Reprezentarea informațiilor

Labirintul este reprezentat ca o matrice `L`, cu `nxm` elemente. Elementele labirintului sunt inițial `0` (semnificând culoar) şi `1` (semnificând perete). Pentru ca şoricelul să nu treacă de mai multe ori prin aceeaşi poziție, existând riscul de a intra în buclă, vom marca în labirint pozițiile prin care trece şoricelul cu `2`.

Pentru a determina pozițiile în care se poate deplasa şoricelul, vom utiliza doi vectori `Dx[8]` şi `Dy[8]`, pe care îi inițializăm cu deplasările pe linie, respectiv pe coloană pentru toate cele 8 direcții posibile de mişcare.

Pentru a nu verifica permanent dacă şoricelul nu a ajuns cumva la marginea labirintului, bordăm labirintul cu perete (două linii şi două coloane cu valoarea `1`).

Condiții interne

Din poziția `(x,y)` şoricelul se poate deplasa pe direcția `dir`, deci în poziția
`(x+Dx[dir], y+Dy[dir])`
dacă şi numai dacă
`L[x+Dx[dir]][y+Dx[dir]]=0`
(această poziție reprezintă un culoar prin care şoricelul nu a mai trecut).

```
#include<iostream>                          #include<fstream>
```

```
using namespace std;
ifstream f("labirint.in");
ofstream g("labirint.out");

#define DimMax 20
int Dx[8]={-1,-1,0,1,1,1,0,-1};
int Dy[8]={0,1, 1,1,0,-1,-1,-1};
int L[DimMax][DimMax];
int n, m, xs, ys, xb, yb, NrSol;

void Citire()
{
f>>n>>m>>xs>>ys>>xb>>yb;
for (int i=1; i<=n; i++)
   for (int j=1; j<=m; j++)
      f>>L[i][j];
f.close();
}
void Bordare()
{              //bordam labirintul cu cate un
perete
for (int i=0; i<=n+1; i++)//perete la stanga si la
dreapta
   L[i][0]=L[i][m+1]=1;
for (int j=0; j<=m+1; j++)          //perete sus si
jos
   L[0][j]=L[n+1][j]=1; }
void Afisare()
{
   g<<"Solutia nr. "<<++NrSol<<endl;
for (int i=1; i<=n; i++)
.
   {for (int j=1; j<=m; j++)
      if (L[i][j] == 2)
         g<<'*';
         else
         g<<L[i][j];
         g<<endl;
   }
}
void Cauta(int x, int y)
{ L[x][y]=2;                  //marchez pozitia x y
if (x == xb && y == yb) Afisare();
else
for (int dir=0; dir<8; dir++)
if (!L[x+Dx[dir]][y+Dy[dir]]) //culoar nevizitat
Cauta(x+Dx[dir], y+Dy[dir]);
L[x][y]=0;
/*la intoarcere sterg marcajul, pentru a putea
explora
acest culoar si in alta varianta*/
}
int main()
{
Citire();
Bordare();
Cauta(xs, ys);
if (!NrSol)
   g<<"Nu exista solutii!\n";
g.close();
return 0;
}
```

Tema:

1. Se citeste un numar natural n. Sa se afiseze toate modurile in care poate fi descompus ca produs de numere naturale diferite de 1 si n.
   Exemplu:
   36 poate fi descompus ca:
   2*2*3*3
   2*2*9
   2*18
   3*3*4
   .....
2. Din fisierul cub.in se citesc de pe prima linie 2 numere naturale n si m si de pe urmatoarele n linii n perechi l si c unde l este lungimea laturii, iar c culoarea pentru n cuburi. l este numar natural, iar c este sir de caractere de lungime maxim 20. Sa se construiasca toate turnurile formate din cel putin m cuburi care se pot forma din cuburile citite din fisier stiind ca un cub se poate pune peste un altul doar daca are latura strict mai mica si culoarea diferita de a celui peste care vrem sa îl punem. Sa se afiseze turnurile obtinute si turnul format din cele mai multe cuburi. Un turn se afiseaza începând cu cel mai de sus cub.
   Exemplu:
   3 2
   3 verde
   4 rosu
   1 rosu

Se obtin turnurile:

1 rosu	3 verde	1 rosu
3 verde	4 rosu	3 verde
4 rosu		

3. Fie n>0, natural. Sa se scrie un program care sa afiseze toate partitiile unui numar natural n. Numim partitie a unui numar natural nenul n o multime de numere naturale nenule {p1, p2, …, pk} care îndeplinesc conditia p1+p2+ …+pk = n.
   Ex: pt n = 4 programul va afisa:
   4 = 1+1+1+1
   4 = 1+1+2
   4 = 1+3
   4 = 2+2
   4 = 4